Plugin JUnit

Mikaël Marche

April 26, 2006

JUnit is a Java API enabling to describe unit tests for a Java application. The JUnit plugin inside Salomé-TMF enables one to execute automatically JUnit tests with the « TestCase » stereotype.

Contents

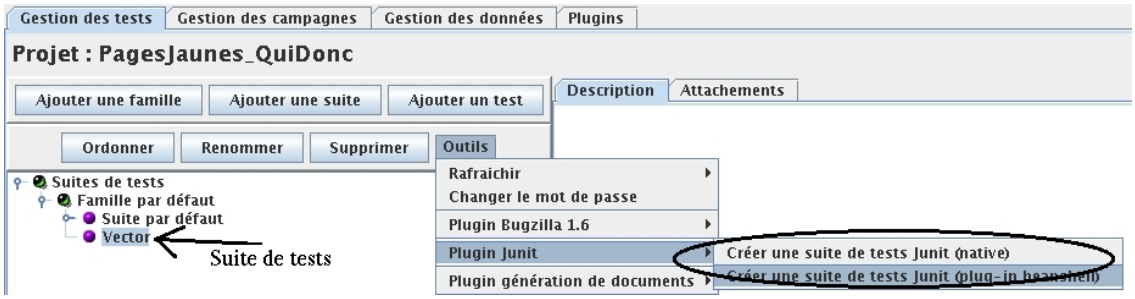
1 Creating a JUnit test suite

The JUnit plugin enables to create a JUnit test suite by starting from tests declared within an object with the « TestCase » stereotype. For this, you need a « .jar » or « .zip » file containing the compiled code for this object.

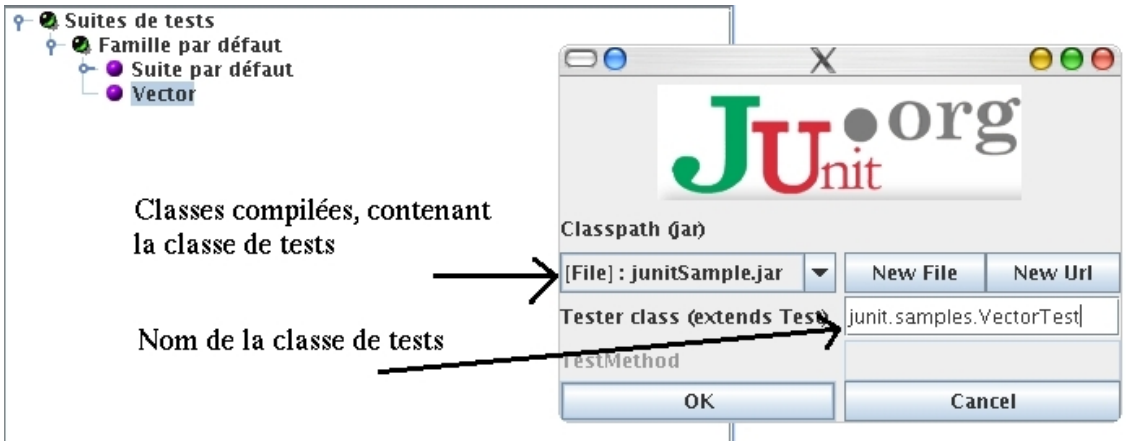
There exists two modes for creating JUnit test suites. A first mode is said native, and the other one is linked to the BeanShell plugin. The native mode uses the Java reflexion for instantiating the test cases, whereas the BeanShell mode produces BeanShell test scripts which describe the instantiation and the execution of the test class.

In the native mode, the « .jar » or « .zip » file must countain the test object with the « TestCase » stereotype, but also the set of classes necessary for executing the tests. This restriction is caused by the fact that, during the execution of the native mode, there can be no linking to an environment or execution script for loading other classes.

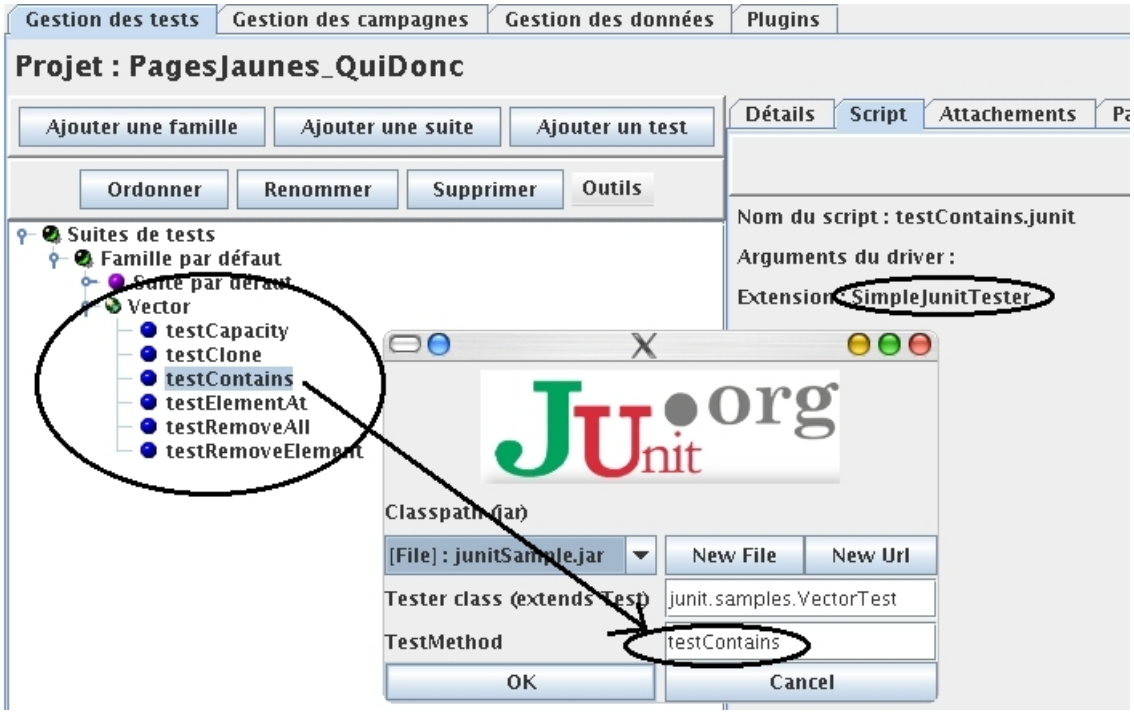
For creating a JUnit test suite, after selecting a test suite in the project, choose the menu « Tools->Plugin junit », and then the feature « Create a JUnit test suite ».



You should then fill the class creation form as follows:



Starting from those informations, the JUnit plugin analyses the selected test class for producing the set of tests contained in this class as automatic test cases inside Salomé (BeanShell or native test cases, depending upon the selected mode).



2 Updating a JUnit test

Updating a JUnit test will depend upon the selected mode when creating the test suite. In native mode, the creation form enables to update the informations « classpath, Tester class, and TestMethod » which uniquely identify the test. In BeanShell mode, it is possible to modify the BeanShell script executing the test class (see the documentation for the BeanShell plugin).

3 Using objects and test parameters

Whatever the creation mode of a JUnit test case, it is possible to use in the JUnit test class the test parameters linked to one execution and the objects of the Salomé GUI (*=org.objectweb.salome_tmf.data).

Name	Class	Description
date	*Java.lang.Date*	Execution date
time	*Java.lang.Time*	Execution time
salome_projectName	*Java.lang.String*	Name of the current project
salome_projectObject	**.Project*	Reference of the project object in Salomé
salome_debug	*boolean*	False when the script evaluates in the editor
salome_ExecResultObject	**.ExecutionResult*	Reference on the current execution results
salome_ExecTestResultObject	**.ExecutionTestResult*	Reference on the execution result of the current test case
salome_CampagneName	*Java.lang.String*	Name of the current campaign
salome_CampagneObject	**.Campaign*	Reference of the current campaign
salome_environmentName	*Java.lang.String*	Name of the current environment
salome_environmentObject	**.Environment*	Reference of the current environment
salome_ExecName	*Java.lang.String*	Name of the current execution
salome_ExecObject	**.Execution*	Reference on the current execution
salome_TestName	*Java.lang.String*	Name of the current test case
salome_TestObject	**.Test*	Reference of the current test case
salome_SuiteTestName	*Java.lang.String*	Name of the current test suite
salome_SuiteTestObject	**.TestList*	Reference of the current test suite
salome_FamilyName	*Java.lang.String*	Name of the current test family
salome_FamilyObject	**.Family*	Reference of the current test family
testLog	*Java.lang.String*	Test log to add as an attachment at the execution

In JUnit native mode, if the test class has got a constructor method which takes as argument a Hashtable, the test class will be instantiated with this constructor method and an Hashtable object which contains all the valued test parameters and the previous objects.

```
import junit.framework.TestCase;
import org.objectweb.salome_tmf.data.Environment;
import java.util.Hashtable;

public class  TestParam extends TestCase{

String url_server_Env;
String Adress_services_schema_Env;
String url_server_DS;
String Adress_services_schema_DS;

    public TestParam(String name) {
                System.out.println("Constructor with String used, arg=" + name);
    }

    public TestParam(Hashtable params) {
                System.out.println("Constructor with Hashtable used, arg=" + params);
                /* sample to salome_tmf object in Java */
                Environment env = (Environment)params.get("salome_environmentObject");

                /* sample to use environment parameters */
                url_server_Env = env.getParameterValue("url_server");
                Adress_services_schema_Env =  env.getParameterValue("Adress_services_schema");

                /* sample to use dataset parameters */
                url_server_DS = (String) params.get("url_server");
                Adress_services_schema_DS = (String) params.get("Adress_services_schema");
    }

    public void testParam() throws Exception {
                System.out.println("(env) url_server = " + url_server_Env);
```

```
                System.out.println("(env) Adress_services_schema = " + Adress_services_schema_Env);

                System.out.println("(DataSet) url_server = " + url_server_DS);
                System.out.println("(DataSet) Adress_services_schema = " + Adress_services_schema_DS);
    }
}
```

In BeanShell mode, the Hashtable assignment is done by calling the method « setParam(Hashtable) » if this method exists in the test class.

```
import junit.framework.TestCase;
import org.objectweb.salome_tmf.data.Environment;
import java.util.Hashtable;

public class  TestParam extends TestCase{

String url_server_Env;
String Adress_services_schema_Env;
String url_server_DS;
String Adress_services_schema_DS;

    public void setParam(Hashtable params) {
                /*sample to salome_tmf object in Java
                Environment env = (Environment)params.get("salome_environmentObject");

                /* sample to use environment parameters */
                url_server_Env = env.getParameterValue("url_server");
                Adress_services_schema_Env =  env.getParameterValue("Adress_services_schema");

                /* sample to use dataset parameters */
                url_server_DS = (String) params.get("url_server");
                Adress_services_schema_DS = (String) params.get("Adress_services_schema");
    }

    public void testParam() throws Exception {
                System.out.println("(env) url_server = " + url_server_Env);
                System.out.println("(env) Adress_services_schema = " + Adress_services_schema_Env);

                System.out.println("(DataSet) url_server = " + url_server_DS);
                System.out.println("(DataSet) Adress_services_schema = " + Adress_services_schema_DS);
    }
}
```

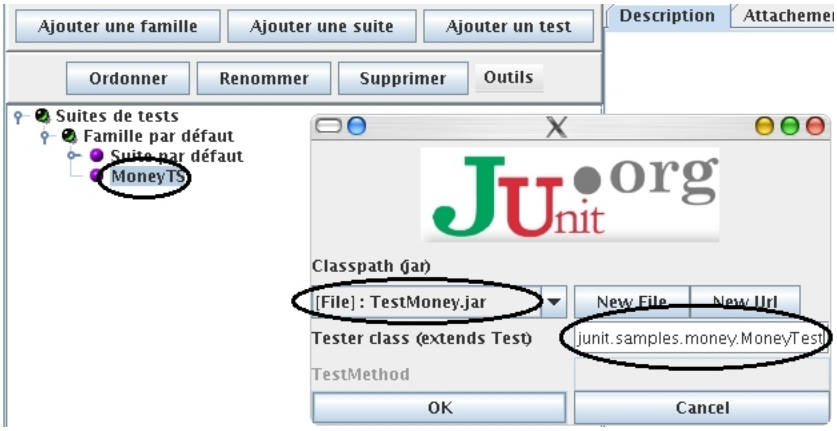
4 Example

The following example demonstrates how to use the JUnit plugin in native and BeanShell mode, starting from a test class distributed in the release 3.8.1 of JUnit. We suppose that you have compiled two « .jar » files, one with the compiled code for the class ń junit.samples.money.MoneyTest ż, and the other one with the compiled code for the class under test called « junit.samples.money.Money ».

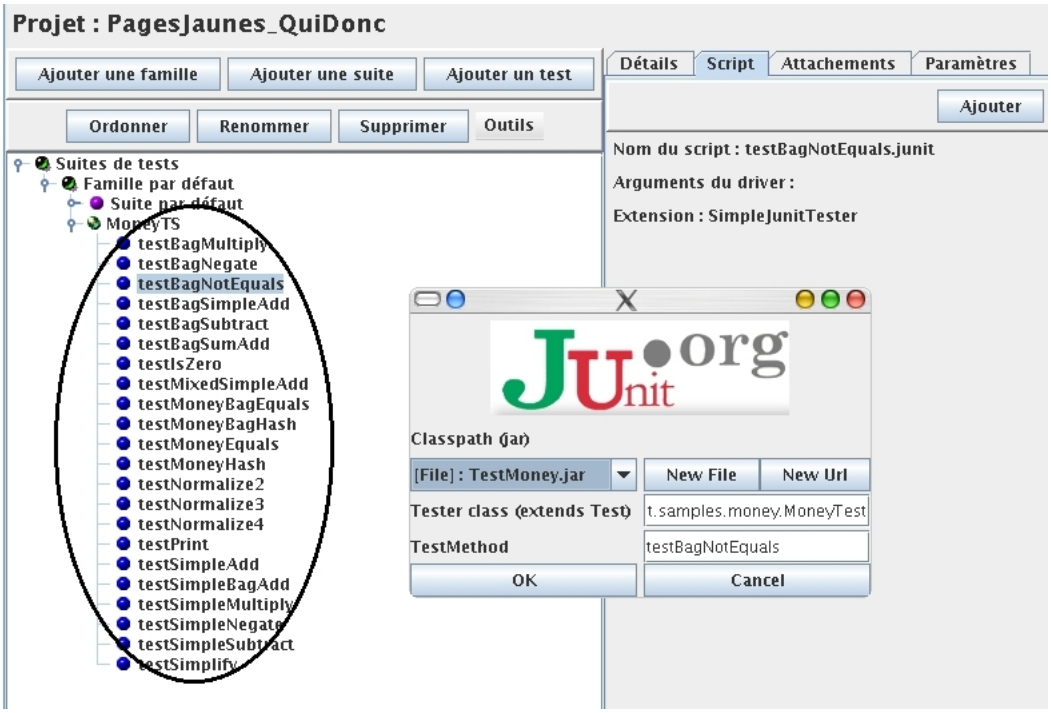
Nota bene: the example uses the environment scripts for loading the classes of the tested object. It is possible not to use the environment scripts if the « .jar » file containing the test cases contains also the classes of the tested object.

4.1 Creation of the JUnit test suite

Let us call ń TestMoney.jar ż the file containing the test class « junit.samples.money.MoneyTest ». We create a JUnit test suite as follows:



The set of tests contained in this class is:

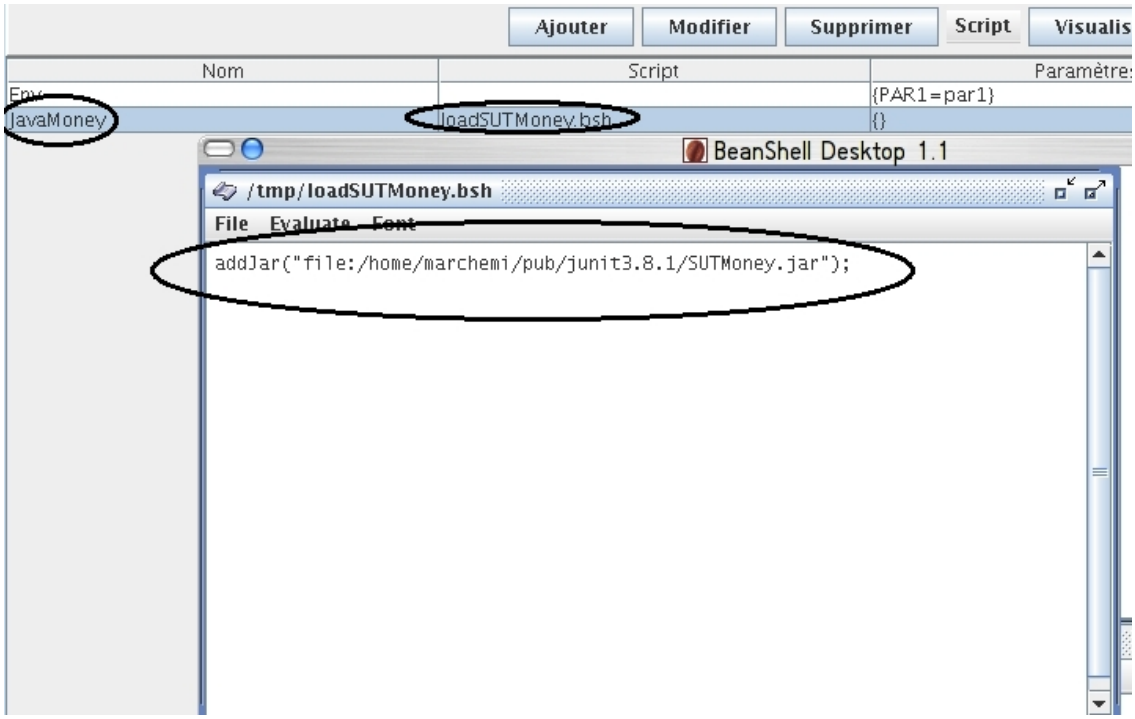


4.2 Creation of the environment under test

Let us call « SUTMoney.jar » the file containing the classes required for executing the tests of the class « junit.samples.money.MoneyTest ». We will now create an environment whose script will load the classes contained in the file « SUTMoney.jar ». This operation is required only when the file « TestMoney.jar » doesn't contain all the classes required for executing the tests.

We suppose that we have created an environment « JavaMoney », and we will add to this environment a BeanShell script which will load the classes of the file « SUTMoney.jar ». For filling this task, the following script will make the job:

```
addJar("file:/home/marchemi/pub/junit3.8.1/SUTMoney.jar");
```



The environment script can load several « .jar » files, and, if needed, can make several BeanShell tasks.

Also, it is possible to update the file *plugin.xml* of the simpleJunit directory for loading a set of Java libraries.

```
<?xml version="1.0" ?>
<!DOCTYPE plugin PUBLIC "-//JPF//Java Plug-in Manifest 0.2" "http://jpf.sourceforge.net/plugin_0_2.dtd">
<plugin id="simpleJunit" version="1" class="salomeTMF_plug.simpleJunit.SimpleJunitPlugin">
        <requires>
                <import plugin-id="core"/>
        </requires>
        <runtime>
                <library id="simpleJunit" path="simpleJunit/simpleJunit.jar" type="code"/>
                <library id="junit" path="simpleJunit/libs/HTTPUnits/junit.jar" type="code"/>
                <library id="httpjunit" path="simpleJunit/libs/HTTPUnits/httpunit.jar" type="code"/>
                <library id="xsdlib" path="simpleJunit/libs/libeXist/xsdlib.jar" type="code"/>
        </runtime>
```

```
        <extension plugin-id="core" point-id="TestDriver" id="simpleJunit.TestDriver">
                <parameter id="class" value="salomeTMF_plug.simpleJunit.SimpleJunitPlugin"/>
                <parameter id="name" value="Junit"/>
                <parameter id="description" value="Plugin Junit pour la definition de classe de tests"/>
                <parameter id="extensions" value=".junit"/>
        </extension>
        <extension plugin-id="core" point-id="Common" id="simpleJunit.Common">
                <parameter id="class" value="salomeTMF_plug.simpleJunit.SimpleJunitPlugin"/>
                <parameter id="name" value="JunitTestSuite"/>
                <parameter id="description" value="Creation de suite HTTPJunit TVMoblie"/>
        </extension>
</plugin>
```

4.3 Running the execution

For executing the test suite, it's enough to create a campaign including the JUnit test cases and to associate an execution with the environment « JavaMoney ».